Terminal Dashboards

Render HTML Dashboards Directly in Your Terminal

APRIL 5, 2026

THEDECIPHERIST.COM

TABLE OF CONTENTS

THE PROBLEM

Every developer monitoring workflow looks the same: write code in your editor, switch to a browser tab to check Grafana, flip to another tab for Docker stats, open a terminal for `htop`, check another tab for your database. Context switching kills focus.

What if your dashboards lived right next to your code — inside the same terminal you're already using?

WHAT TUIMON DOES

TuiMon renders HTML pages directly in your terminal. It takes any HTML file — complete with CSS flexbox, grid layouts, Chart.js charts, D3 visualizations — runs it in a headless Chromium browser, captures a PNG screenshot, and streams it to your terminal using modern graphics protocols (Kitty, Sixel, or iTerm2).

The result: pixel-perfect dashboards that update in real time, displayed right inside your terminal emulator. If you're using VSCode, that means your dashboard renders in the integrated terminal panel while your code sits in the editor above it. You never leave your IDE.

A typical frame renders in about 50ms from data push to pixels on screen.

THREE LEVELS OF USAGE

TuiMon is designed so that the simplest use case requires zero configuration, while the most advanced use case gives you full control over every pixel.

Level 1: Zero Config — Point at Data

The fastest way to use TuiMon is to point it at something:

```
tuimon data.json          # JSON → instant table + charts
tuimon sales.csv          # CSV → auto-detected trends
tuimon access.log         # nginx → request stats dashboard
tuimon modsec_audit.log   # ModSecurity → attack dashboard
tuimon results.xml        # JUnit → test results
tuimon coverage.json      # Istanbul → coverage report
```

TuiMon auto-detects the file format, parses the data, infers the best layout (tables for tabular data, line charts for time series, doughnut charts for distributions), and renders a dashboard. It also watches the file for changes, so if something writes to that log file, your dashboard updates automatically.

Press `D` to switch to a full-screen data table. Arrow keys page through the data. ESC goes back. It's instant.

Built-in Presets

For common developer tasks, TuiMon ships with ready-made dashboards:

Docker — `tuimon docker` shows live container stats with CPU and memory line charts that build up over time, a container details table showing Net I/O, Block I/O, memory in MB/GB, and PIDs. Press `L` to switch to a full container logs viewer.

Git — `tuimon git` analyzes your repository: commit frequency over the last 30 days, top contributors, most changed files with change counts, and a scrollable recent commit log.

Process Monitor — `tuimon ps` shows real-time CPU and memory usage per process, load averages, CPU history, and a full process table with PID, user, CPU%, memory%, and command. Press `P` for a dedicated process table page.

Dependencies — `tuimon package-lock.json` (or `yarn.lock`) parses your lock file and shows total dependencies, direct vs dev split, duplicates, version conflicts, and a browsable dependency table with a "most depended on" chart.

Test Coverage — `tuimon coverage.json` parses Istanbul JSON, lcov, or JUnit XML and shows coverage percentages, file-level breakdown, and highlights low-coverage files.

Watch Live Data

You can also watch a JavaScript module or an HTTP endpoint:

```
tuimon watch metrics.js                          # calls your exported functior
tuimon watch --url http://localhost:3000/metrics  # polls a JSON endpoint
```

For the JS module approach, just export a function that returns your data:

```
module.exports = () => ({
  cpu: getCpuPercent(),
  memory: getMemPercent(),
  uptime: process.uptime(),
})
```

TuiMon calls your function on interval, auto-detects the data shape, and builds a dashboard. No configuration needed.

Level 2: Declarative — Define Widgets in TypeScript

When you want more control over the layout but don't want to write HTML, use the declarative API. Define your widgets in TypeScript and TuiMon generates the dashboard with its built-in dark neon theme:

```
import tuimon from 'tuimon'

const dash = await tuimon.start({
  pages: {
    main: {
      default: true,
      layout: {
        title: 'Production API',
        stats: [
          { id: 'users', label: 'Users Online', type: 'stat' },
          { id: 'cpu', label: 'CPU', type: 'gauge' },
          { id: 'memory', label: 'Memory', type: 'gauge' },
        ],
        panels: [
          { id: 'traffic', label: 'Request Traffic', type: 'line', span: 2 },
          { id: 'health', label: 'Service Health', type: 'status-grid' },
          { id: 'endpoints', label: 'Endpoints', type: 'doughnut' },
          { id: 'events', label: 'Recent Events', type: 'event-log', throttle:
          { id: 'methods', label: 'HTTP Methods', type: 'bar' },
        ],
      },
    },
  },
  refresh: 500,
  data: () => ({
    users: 1847,
    cpu: 67,
    memory: 54,
    traffic: { Requests: 312, Errors: 10, 'Latency (ms)': 80 },
    health: ['API Gateway', 'Auth Service', 'Database', 'Cache', 'CDN'],
    endpoints: { '/api/users': 35, '/api/orders': 25, '/api/auth': 20 },
    events: ['Deploy v2.4.1 completed', 'Cache cleared — 2.3GB freed'],
    methods: { GET: 220, POST: 85, PUT: 35, DELETE: 15 },
  }),
})
```

Eight Widget Types

TYPE	WHAT IT SHOWS	DATA FORMAT
`stat`	Single number or text	`42` or `'2h 15m'`
`gauge`	0-100% progress bar	`73`
`line`	Time-series line chart	`{ Series1: 340, Series2: 12 }`
`bar`	Categorical bar chart	`{ GET: 220, POST: 85 }`
`doughnut`	Proportion chart	`{ Users: 35, Orders: 25 }`
`event-log`	Scrolling timestamped events	`['Deploy completed']`
`status-grid`	Health indicator dots	`['Node-1', 'Node-2']`
`table`	Paginated data table	`[{ name: 'Alice', role: 'Admin' }]`

Smart Data Normalization

You don't need to learn a special data format. Send the simplest thing and TuiMon figures out the rest:

- Send `42` to a stat widget → it shows `42`
- Send `73` to a gauge widget → it shows a 73% progress bar, color-coded (green < 60, amber < 85, red ≥ 85)
- Send `{ Requests: 340, Errors: 12 }` to a line widget → it auto-accumulates history, so each render call adds a data point and the chart grows over time
- Send `['Deploy completed']` to an event-log → it adds a timestamp and appends to the scrolling list

Per-Widget Throttle

Each widget can update at its own speed. Your line charts can update every frame (every 500ms in the example above) while your event log throttles to every 2 seconds and your status grid to every 5 seconds:

```
panels: [
  { id: 'chart', type: 'line' },                        // every frame
  { id: 'events', type: 'event-log', throttle: 2000 },  // max every 2s
  { id: 'health', type: 'status-grid', throttle: 5000 },// max every 5s
]
```

Level 3: Full HTML — Total Control

This is what TuiMon was originally built for. You write your dashboard as a standard HTML page using any web technology you want — CSS flexbox, grid, animations, Chart.js, D3, Three.js, whatever runs in Chromium.

TuiMon auto-injects a small client script into your page that provides the `TuiMon` API:

```
// Receive live data
TuiMon.onUpdate(function(data) {
  document.getElementById('cpu').textContent = data.cpu + '%'
  updateMyChart(data.traffic)
})

// Shortcut: set element content by selector
TuiMon.set('#memory', '54%')
TuiMon.set('#status', { textContent: 'OK', style: 'color: green' })

// Trigger a notification in the F-key bar
TuiMon.notify('Data exported!', 3000)
```

Multiple Pages with Keyboard Navigation

Define multiple HTML pages and let users switch between them with single-letter keyboard shortcuts:

```
pages: {
  overview: { html: './pages/overview.html', default: true },
  cpu:      { html: './pages/cpu-detail.html', shortcut: 'g', label: 'CPU Detai
  memory:   { html: './pages/memory-detail.html', shortcut: 'm', label: 'Memory
  network:  { html: './pages/network.html', shortcut: 'n', label: 'Network' },
}
```

Press `g` to jump to CPU detail, `m` for memory, `ESC` to go back to the overview. The F-key bar at the bottom always shows the current page's available bindings.

F-Key Bindings

Each page can define its own F-key actions:

```
keys: {
  F1:  { label: 'Help',    action: () => showHelp() },
  F5:  { label: 'Refresh', action: async () => dash.render(await getData()) },
  F9:  { label: 'Export',  action: () => exportCSV() },
  F10: { label: 'Quit',    action: () => process.exit(0) },
}
```

Shortcut Badges

Add `data-tm-key="g"` to any HTML element and TuiMon automatically renders a `[G]` badge in the corner, so users immediately know they can press G to navigate there.

DATABASE VIEWER

TuiMon includes a built-in database viewer that works without installing any new dependencies. It detects your project's existing database driver from `node_modules/` and reads the connection string from `.env`.

```
tuimon db users                              # view a table/collection
tuimon db "SELECT * FROM users WHERE active"  # custom SQL query
tuimon db users --query '{"active": true}'   # MongoDB filter
tuimon db users --watch                      # re-query every 2 seconds
tuimon db users --watch --interval 5000      # custom poll interval
tuimon db users -c "name,email,role"         # show specific columns
tuimon db users --uri "mongodb://..."        # explicit connection string
tuimon db users --env MY_DB_URI              # use a specific env variable
```

Supports MongoDB (`mongodb`), PostgreSQL (`pg`), MySQL (`mysql2`), and SQLite (`better-sqlite3` or `sqlite3`).

TERMINAL SUPPORT

TuiMon auto-detects your terminal's graphics protocol. No configuration needed.

TERMINAL	PROTOCOL	STATUS
Kitty	Kitty	Fully supported
Ghostty	Kitty	Fully supported
WezTerm	Kitty	Fully supported
iTerm2	iTerm2	Fully supported
VSCode	Sixel	Fully supported
mlterm	Sixel	Fully supported

For VSCode, run `tuimon init` to automatically enable `terminal.integrated.enableImages: true` in your workspace settings. Or add it manually.

Run `tuimon check` to verify your terminal supports graphics protocols and see the detected pixel dimensions.

ARCHITECTURE

TuiMon's render pipeline:

1. **Your data function** returns an object with the current values
2. **TuiMon pushes the data** into the browser page via `window.__tuimon_update__(data)`
3. **The page updates** – your `TuiMon.onUpdate()` callback fires, charts redraw, DOM updates
4. **After a short delay** (configurable, default 50ms), TuiMon takes a PNG screenshot
5. **The encoder** converts the PNG to the appropriate terminal graphics format – base64 chunks for Kitty, raw pixel bands for Sixel

6. **The encoded image** is written directly to stdout as terminal escape sequences
7. **Your terminal** renders the image inline

The F-key bar, keyboard input, and navigation state machine all run natively in Node.js using raw stdin mode — they don't go through the browser at all.

GETTING STARTED

```
# Install globally
npm install -g tuimon

# Check terminal support
tuimon check

# Scaffold a starter project
tuimon init
tuimon start

# Or just point at data
tuimon data.json
tuimon docker
tuimon git
tuimon ps
```

LINKS

- Documentation Site
- GitHub
- npm